Autoassociative Neural Network

Presented by:

Yevgeniy Gershteyn
Larisa Perman

04/17/2003

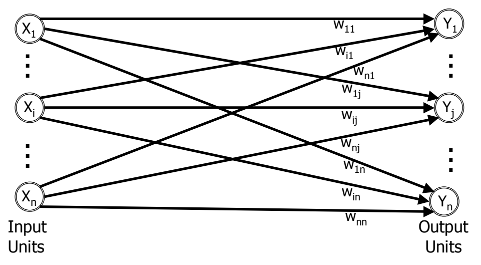
Definition

- Autoassociative neural networks (AANN) are feedforward networks whose input and output vectors are identical.
- The process of training is called *storing* the vectors (binary or bipolar).
- A stored vector can be retrieved from distorted or noisy input, if the input is sufficiently similar to it.
- AANNs are typically used for tasks involving pattern completion.
- The performance of the net is derived from its ability to reproduce a stored pattern from noisy input.

Definition (cont)

- AANNs are special kinds of neural networks that are used to simulate (and explore) associative processes.
- Association in these types of neural networks is achieved through the interaction of a set of simple processing elements (called *units*), which are connected through *weighted connections.*

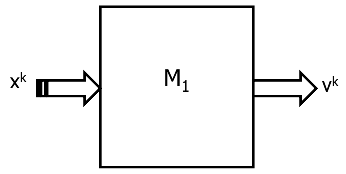
Architecture



Autoassociative neural net

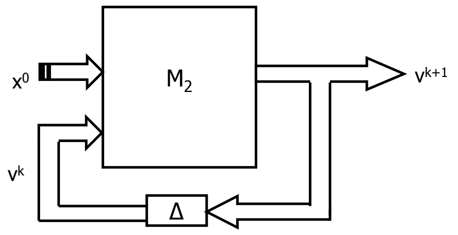
Types of AANN

- Linear (feedforward) AANN



Types of AANN (cont)

- Recurrent AANN



Bottleneck Feature

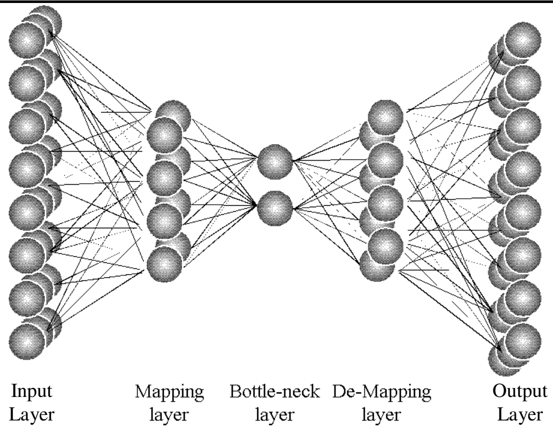
- The key feature of such networks is the data compression performed by the bottleneck layer; this provides the topology with very powerful properties of feature extraction.
- The network consists of an input layer followed by a non-linear hidden layer (the bottleneck layer), a second non-linear hidden layer and finally the output layer of the same dimension as the input.

7

AANN Example

- AANN is a five-layer perceptron feedforward network which can be viewed as two independent three-layer neural networks connected in series.
- The first network mixes and compresses the ***n*** redundant measurements into a smaller number of characteristic variables which should ideally represent the essential characteristics of the process.
- The second network works in the opposite way and uses the compressed information to regenerate the original ***n*** redundant measurements.

8



Algorithm

- For mutually orthogonal vectors, the Hebb rule can be used for setting the weights because the input and output vectors are perfectly correlated, component by component (Same number of output units as input units)
- General rules:
  - When unit A and B are simultaneously correlated, increase the strength of the connection between them.
  - When unit A and B are counter-correlated, decrease the strength of the connection between them.
- Hebbian Learning was introduced by Dr. Reznik in topic 6.

10

Algorithm (cont)

- Step 0.
  - Initialize all weights, i = 1, . . . , n; j = 1, . . . , n: $\mathbf{w_{ij} = 0}$;
- Step 1.
  - For each vector to be stored, do Steps 2 – 4:
- Step 2.
  - Set activation for each input unit, i = 1, . . . , n: $\mathbf{x_i = s_i}$;
- Step 3.
  - Set activation for each output unit, j = 1, . . . , n: $\mathbf{y_j = s_j}$;
- Step 4.
  - Adjust the weights, i = 1, . . . , n; j = 1, . . . , n:
    $\mathbf{w_{ij}(new) = w_{ij}(old) + x_i y_j}$.

11

Algorithm (cont)

- In practice usually do not use the algorithmic form of Hebb learning, and the weights usually set from the formula:

$$\boxed{\mathbf{W} = \sum_{p=1}^{P} \mathbf{s}^{\mathbf{T}}(p)\, \mathbf{s}(p)}$$

Where:

W – weighted matrix

S(p) – P distinct n-dimensional prototype patterns

T – learning step in time t (from Hebbian learning)

12

Application

- AANN can be used to determine whether an input vector is "known" or "unknown".
- AANN recognizes a "known" vector by producing a pattern of activation on the output unit of the net that is the same as one of the vectors stored in it.
- The application procedure for bipolar inputs and activations:
- Step 0.
  - Set the weights using Hebb rule (outer product)

Application (cont)

- Step 1.
  - For each testing input vector, do Steps 2 – 4.
- Step 2.
  - Set activations of the input units equal to the input vector.
- Step 3.
  - Compute net input to each output unit, j = 1, . . . , n:

$$\mathbf{y\_in_j} = \sum_i \mathbf{x_i\, w_{ij}}.$$

- Step 4.
  - Apply activation function (j = 1, . . . , n):

$$\mathbf{y_j} = \boldsymbol{f}\,(\mathbf{y\_in_j}) = \begin{cases} 1 & \text{if } y\_in_j > 0; \\ -1 & \text{if } y\_in_j \le 0. \end{cases}$$

Example 1

- *Store one pattern (vector) in AANN and then recognize it.*
- Step 0.
  - Vector s = (1, 1, 1, -1) is stored with the weight matrix:

$$W = \begin{bmatrix} 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & -1 \\ -1 & -1 & -1 & 1 \end{bmatrix}$$

- Step 1.
  - For each testing input vector, do Steps 2 – 4.

Example 1 (cont)

- Step 2.
  - x = (1, 1, 1, -1).
- Step 3.
  - y_in = (4, 4, 4, -4).
- Step 4.
  - y = f(4, 4, 4, -4) = (1, 1, 1, -1).
- So, the input vector is recognized as a "known" since the response vector y is the same as the stored vector.

$(1, 1, 1, -1)\cdot W = (4, 4, 4, -4) \rightarrow (1, 1, 1, -1)$

Example 2

- *Testing an AANN: one "mistake" in the input vector.*
- Each vector **x** is formed from the original stored vector **s** (see Example 1) with a "mistake" in one component.

  $(-1, 1, 1, -1) * W = (2, 2, 2, -2) \rightarrow (1, 1, 1, -1)$
  $(1, -1, 1, -1) * W = (2, 2, 2, -2) \rightarrow (1, 1, 1, -1)$
  $(1, 1, -1, -1) * W = (2, 2, 2, -2) \rightarrow (1, 1, 1, -1)$
  $(1, 1, 1, 1) * W = (2, 2, 2, -2) \rightarrow (1, 1, 1, -1)$

- *Testing an AANN: two "mistakes" in the input vector.*

  $(-1, -1, 1, -1) * W = (0, 0, 0, 0)$

  The net does not recognize this input vector

Example 3

- *Testing an AANN: two "missing" entries in the input vector.*
- Each vector **x** is formed from the original stored vector **s** (see Example 1) with two "missing" data components.

  $(0, 0, 1, -1) * W = (2, 2, 2, -2) \rightarrow (1, 1, 1, -1)$
  $(0, 1, 0, -1) * W = (2, 2, 2, -2) \rightarrow (1, 1, 1, -1)$
  $(0, 1, 1, 0) * W = (2, 2, 2, -2) \rightarrow (1, 1, 1, -1)$
  $(1, 0, 0, -1) * W = (2, 2, 2, -2) \rightarrow (1, 1, 1, -1)$
  $(1, 0, 1, 0) * W = (2, 2, 2, -2) \rightarrow (1, 1, 1, -1)$
  $(1, 1, 0, 0) * W = (2, 2, 2, -2) \rightarrow (1, 1, 1, -1)$

Example 4

- *An AANN with no self-connections: zeroing-out the diagonal.*

$$W_0 = \begin{bmatrix} 0 & 1 & 1 & -1 \\ 1 & 0 & 1 & -1 \\ 1 & 1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix}$$

- Each vector x is formed from the original stored vector s (see Example 1) with two "mistakes" in first and second components.

  $(-1, -1, 1, -1) * W_0 = (-1, 1, -1, 1)$

  The net still does not recognize this input vector

19

Example 4 (cont)

- If the weight matrix W_0 is used for the case of "missing" components in input data, the net recognizes each of these input vectors whose input components were zero.

$(0, 0, 1, -1) * W_0 = (2, 2, 1, -1) \rightarrow (1, 1, 1, -1)$
$(0, 1, 0, -1) * W_0 = (2, 1, 2, -1) \rightarrow (1, 1, 1, -1)$
$(0, 1, 1, 0) * W_0 = (2, 1, 1, -2) \rightarrow (1, 1, 1, -1)$
$(1, 0, 0, -1) * W_0 = (1, 2, 2, -1) \rightarrow (1, 1, 1, -1)$
$(1, 0, 1, 0) * W_0 = (1, 2, 1, -2) \rightarrow (1, 1, 1, -1)$
$(1, 1, 0, 0) * W_0 = (1, 1, 2, -2) \rightarrow (1, 1, 1, -1)$

20

Storage Capacity

- One of the main features of the AANN is the number of patterns or pattern pairs that can be stored before the net begins to forget.
- The number of vectors that can be stored in the net is called the *capacity* of the net.
- The capacity of the AANN depends on the number of components the stored vectors have and the relationships among the stored vectors; more vectors can be stored if they are mutually orthogonal.
- Generally, n – 1 mutually orthogonal bipolar vectors, each with *n* components, can be stored using the sum of the outer product weight matrices (zero in diagonal).

21

Field of application of AANN

- Pattern Recognition
- Voice Recognition
- Bioinformatics
- Signal Validation
- Net Clustering
- etc.

22

Example: Clustering on the Net

- AANN is trained on all 3000 coded messages (from over 30 newsgroups into 46 categories) and then used to construct typical messages under certain specified conditions.
- The recoding resulted in 149 binary features in the new database.
- Applying an AANN to 3000 Messages: Since the data consisted of 149 features, each taking a value of either "0" or "1" after processing, the network has 149 binary units. This leads to 149*149=22,201 weights and 149 thresholds to adjust during training (1 – on, 0 – off).

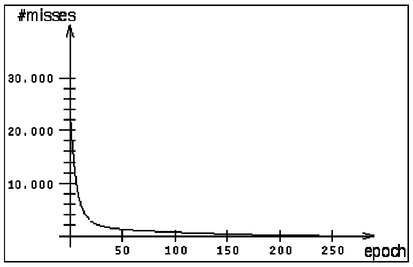
23

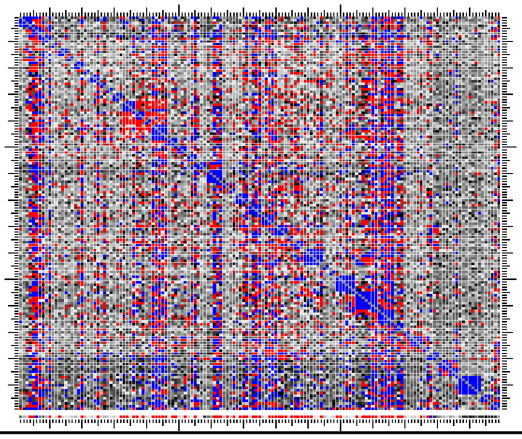
Example: Clustering on the Net

- ***Training a network with over 20,000 weights.***
- Training this neural network obviously requires millions of computations.
  - The network consists of 22,350 parameters
    - 22,201 weights
    - 149 threshold values
  - To adjust and to update each of them, 149 activations have to be computed, each of them requiring again 150 weighted summations.
  - This has to be done for each of the 3,000 patterns in the database, which leads to a computation of over 65 million weighted summations (or connections) plus roughly the same amount of compare and update operations per epoch.
- Almost 100 hours of CPU-time were spent before the error-rate of the network started to settle down after five days.

24

Example: Clustering on the Net





References

- L.Fausett, *Fundamentals of Neural Networks,* Prentice Hall, 1994
- *Auto-associative neural networks,* http://qbab.aber.ac.uk/roy/koho/aaanns.htm
- M. Berthold, F. Sudweeks, S. Newton, R. Coyne *Clustering on the Nets: Applying an Autoassociative Neural Network to Computer-Mediated Discussions,* http://www.ascusc.org/jcmc/vol2/issue4/berthold.html